

EXHIBIT 1

Ser. No.: 10/005,728

6

42390P5943C



ARTICLES BYTEMARKS FACTS

Search BYTE.com

Write to Byte
Editorial Calendar

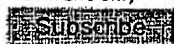
Categories
Previous Editions
Columns
Features

Print Archives
1994-1998

About Us
Byte Editorial Staff
Advertise with Byte
Privacy Policy

Free E-mail Newsletter
from BYTE.com

☒ Byte.com Update

☒ Text only


Building the Virtual PC

November 1997 / Core Technologies / Building the Virtual PC

A software emulator shows that the PowerPC can emulate another computer, down to its very hardware

Eric Traut

Development of Virtual PC -- Connectix Corporation's Macintosh application that emulates a PC and its peripherals -- began almost two years ago, in October 1995. The goal from the beginning was to create a fully Intel-compatible PC in software. The effort centered around a core Pentium instruction-set emulator, complete with MMX instructions. True PC emulation also required the reverse-engineering and development of a dozen other PC motherboard devices, including modern peripherals such as an accelerated SVGA card, an Ethernet controller, a Sound Blaster Pro sound card, IDE/ATAPI controller, and PCI bridge interface. The strategy of hardware-level emulation resulted in an application that allows Macintosh users to run not only Windows programs and DOS games but several x86-based OSes, including Windows 95, NT, and NeXT OpenStep.

Pentium Emulation

The heart of Virtual PC is the Pentium recompiling emulator, a sophisticated piece of software written entirely in hand-coded PowerPC assembly language. It translates Pentium instruction sequences into a set of optimized PowerPC instructions that perform the same operation. Translation occurs on a "basic block" basis, where a basic block consists of a sequence of decoded x86 instructions that end on an instruction that abruptly changes the flow of execution (typically a jump, call, or return-from-subroutine instruction). As the recompiler decodes instructions, it analyzes them for "condition code" usage. Finally, it generates a block of PowerPC code that accomplishes the same task. For more details on the process, see ["Virtual PC Operation"](#).

For purposes of speeding things up, the emulator employs the following tricks:

Translation cache: Even though written in PowerPC assembly language, the translator still requires substantial time to generate optimized instruction translations. To reduce this overhead, the emulator caches blocks of translated instructions.

Interinstruction optimization: Because the Pentium is a CISC processor, instructions perform more than one operation. For example, the ADD instruction not only adds two values together, it also produces a number of condition-code flags that tell programs whether the addition produced a zero or negative result. Such codes are used, for example, to determine if a program performs a conditional jump. Most of the time these codes are ignored. The translator analyzes block x86 instructions to determine which flags the program uses (if any). It then generates PowerPC code for those flags actually used. The first two listings "Translated Code" show how one Pentium instruction translates into three PowerPC instructions, while three Pentium instructions can be optimized from into five PowerPC instructions.

Address translation: One of the most difficult Pentium features to emulate is the built-in memory management unit (MMU). This hardware translates linear (logical) addresses into physical memory addresses. Operating systems use the MMU to implement virtual memory and memory protection. Because of the Pentium's small register file, about three in four Pentium instructions reference memory in one way or another. Each memory address potentially needs to be translated before the emulator loads from, or stores to, the referenced address. An MMU implemented in software would impose a high overhead, which would degrade performance. Luckily, this overhead can be avoided: The Connectix engineers were able to program the PowerPC's MMU to mimic the Pentium MMU's behavior, thus managing the address translations in hardware. The Pentium's memory page attributes can also be mirrored in the PowerPC's MMU. For example, if Virtual PC's emulated OS marks a memory page as write-protected, the page mappings are modified so the corresponding PowerPC page is write-protected.

Segment bounds checking: The Pentium architecture includes the archaic concept of memory segments. Every memory reference, such as instruction fetches, operations, loads, and stores, has an associated memory segment. When a segment's bounds are exceeded, the Pentium's MMU generates a general protection fault (GPF). The OS uses GPFs for more than detecting bugs in applications; it can enable a program to "thunk" down into privileged driver-level code not accessible at the application level. Therefore, the Pentium emulator must detect segment-bound faults where appropriate. Although the PowerPC does not contain segmentation hardware akin to the Pentium, Connectix used PowerPC trap instructions to perform segment bounds checks with little or no overhead.

Hardware Emulation

Besides the Pentium processor, a typical PC motherboard contains a dozen or more chips that work together concurrently. All these chips need to be emulated faithfully for compatibility. The Intel architecture provides an I/O address space that's used to access hardware outside of the CPU. You work with this "I/O space" through two instructions -- IN and OUT. When using these instructions, software must specify an I/O port (or address). Virtual PC routes I/O accesses to code modules that emulate each chip. For example, if Virtual PC encounters an IN instruction referencing port 0x21, it calls a routine in the interrupt-controller emulation module that returns the current interrupt mask. Similar module calls

occur for every I/O space access, as the third listing in "Translated Code" sh

Many of the extra chips on a PC motherboard control I/O devices such as the drive, CD-ROM, keyboard, and mouse. For compatibility with the Mac OS Macintosh hardware, Virtual PC performs all I/O through the standard Mac drivers. So, a request sent to the emulated PC's IDE controller to read a sector from the hard drive gets translated into a read operation that's sent to the Mac OS driver.

The most difficult hardware components to emulate involve precise timing. For example, sound is a real-time operation, and any timing perturbation results in clicks or pops as digitally sampled data fails to arrive on time. Because Virt is hosted on the Mac OS (which gives time to other Mac programs running concurrently, as well as Virtual PC), and it needs to emulate several dozen I/O chips in parallel, precise timing isn't always possible. Virtual PC compensates by placing the highest priority on tasks that directly affect the user, such as sound and video.

Performance

Emulated systems are naturally going to be slower than real hardware. But Connectix engineers concentrated on tuning aspects of the emulated hardware required to run popular PC games and productivity applications at a usable performance level. This was especially challenging given that the PowerPC processor emulates not only the Pentium but all the other chips on a PC motherboard.

Performance of Virtual PC is also greatly affected by the host hardware system. The latest PowerPC processors with high clock rates and large on-chip caches run it best. The speed and size of the system's L2 cache is also critical because of the code expansion that occurs during the translation process.

While users will take a performance hit because this is an emulator, Virtual PC successfully emulates the entire PC at a very low level. PC programs -- applications, device drivers, and operating systems alike -- cannot tell they are running on actual PC hardware.

Translated Code

Translation of Single Pentium Instruction

Pentium instruction	PowerPC instructions	
ADD EAX, 20	li	rTemp1, 20
	addco.	PF, rTemp1, rEAX
	mr	rEAX, rPF

Translation of Pentium Instruction Block

Pentium instructions

```
ADD EAX, 20
ADD EBX, 30
ADD ECX, 40
```

PowerPC instructions

```
add      rEAX, rEAX, 20
add      rEBX, rEBX, 30
li       rTemp1, 40
addco.   rPF, rTemp1, rE
mr       rECX, rPF
```

Code Translation for Pentium I/O Instructions

Pentium instructions

```
MOV AL, 8
MOV DX, 0x1F0
OUT DX, AL
AD
D DX, 7
IN AL, DX
RET
```

PowerPC instructions

```
li       rAL, 8
li       rDX, 0x1F0
bl       HandleIDEPort
addi     rDX, rDX, 7
bl       HandleIDEPort
addi     rIP, rIP, 8
b        DispatchToNex
```

Virtual PC Operation

[illustration link \(24 Kbytes\)](#)



Eric Traut (<mailto:traut@connectix.com>) is lead engineer for Virtual PC at Con Apple Computer, he wrote the 680x0 dynamic recompiling emulator for PowerPC



Up Level



Previous



Next

BYTE TOP OF PAGE

Copyright © 2003 CMP Media LLC, Privacy Policy, Terms of Service

Site comments: webmaster@byte.com

SDMG Web Sites: Byte.com, C/C++ Users Journal, Dr. Dobb's Journal, MSDN Magazine, New Architect, SD Expo, SD Magazine, Sys Admin, The Perl Journal, UnixReview.com, Windows Developer Network